PoketCuddle 개발문서

Index

1. 게임 기획 및 소개

기획 내용

땅따먹기와 포켓몬과의 배틀을 컨셉으로, 포켓몬의 공격을 회피 및 방어 하면서 보유한 포켓몬의 스킬을 활용하여 80%이상의 땅을 모두 점령하면 승리하는 게임이다. 땅을 점령하게 되면 적으로 있던 포켓몬을 보유하게 된다. 보유 시 해당 포켓몬의 고유 스킬을 사용할 수 있다.

게임소개

땅의 구역은 점령 된 땅과 점령 중인 땅으로 되어 있다. 게임의 객체로는 크게 점령해야 하는 땅, 플레이어, 포켓몬, 몬스터가 있다.

플레이어

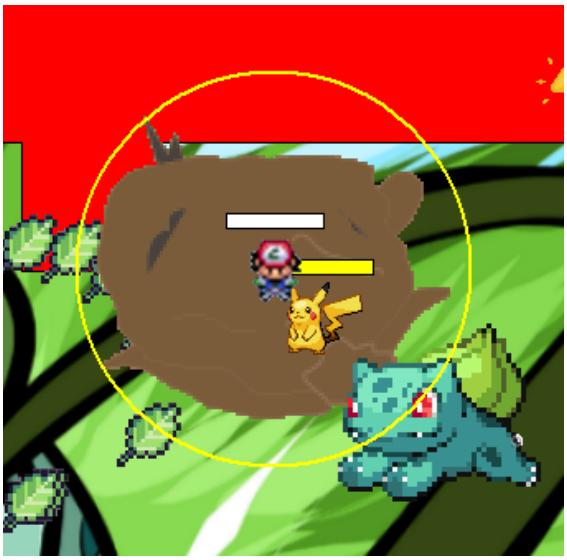
목숨 4개가 주어진다. 좌측 Shift키를 눌러 달리기를 할 수 있으며 달리기 지속시간인 체력이 존재한다. 체력은 플레이어 머리 위에 위치하고 있으며, 하얀색과 파란색으로 표시된다. 체력은 플레이어 위치(점령 된 땅과 점령 중인 땅)에 따라 소모 속도가 상이하고 충전전 되는 경우가 다르다. 점령 된 땅에 위치했을 시 플레이어가 달리기 시 체력은 느리게 소모되며, 걷기 상태에서 이동하여도 체력이 충전된다. 또한 체력을 모두 다 사용하더라도 달리기가 가능하다. 반대로 점령 중인 상태일 때는 체력 소모가 2배 빨라지며, 걷기 상태에서는 체력이 충전되지 않고 오직 멈춘 상태에서만 충전된다.

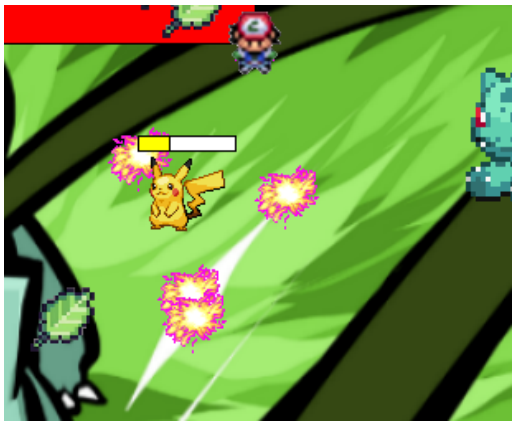
*플레이어 체력 표시 (파란색: 사용된 체력, 하얀색: 남은 체력)

포켓몬 (보유한 포켓몬)

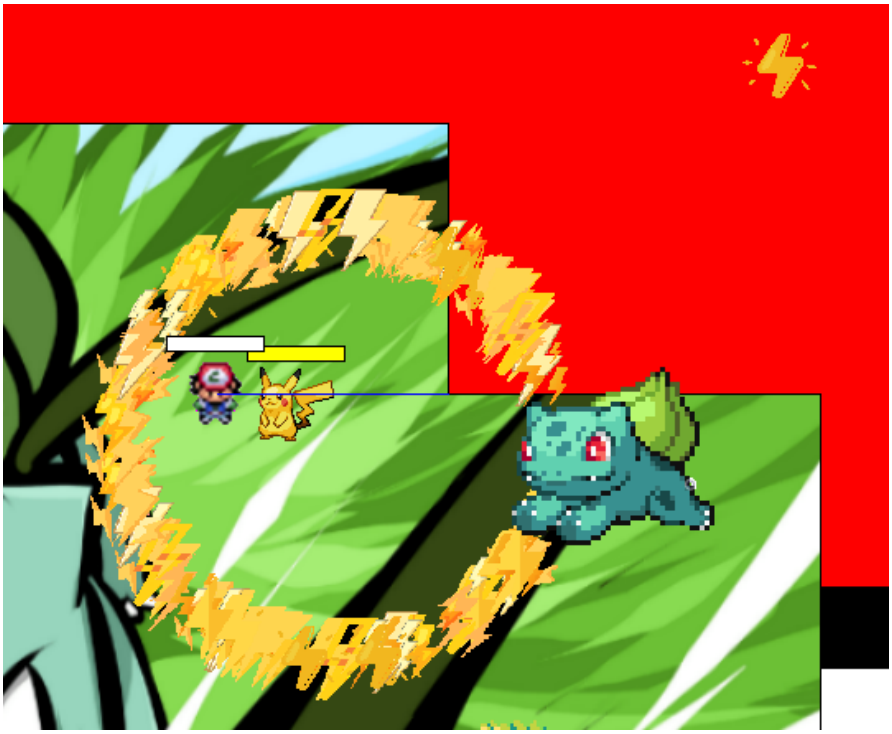
플레이어가 스테이지를 클리어 하게 되면 싸웠던 포켓몬을 보유하게 된다. 포켓몬을 보유하게 되면 해당 포켓몬의 고유 스킬을 사용할 수 있다. 포켓몬은 목숨이나 Collision이 따로 없으며 플레이어를 쫓아 다닌다.

피카츄는 기본으로 제공되는 보유한 포켓몬으로, 썬더볼트와 100만볼트를 활용하여 2가지 스킬을 사용할 수 있다. 썬더볼트는 피카츄를 중심으로 투사체를 발사한다. 해당 투사체는 몬스터와 몬스터의 투사체와 상호작용이 일어난다. 몬스터 같은 경우 투사체를 맞으면 이동 속도가 감소하고, 제한된 시간안에 5번을 맞추면 몬스터는 마비상태가 되어 움직이지 않는다. 몬스터의 투사체와 Collision된 경우는 포켓몬의 투사체와 몬스터의 투사체가 모두 없어진다. 썬더볼트는 Q를 Hold한 상태에서 발생되며, 플레이어 중심으로 생성된 원 안에서 피카츄를 이동시킬 수 있으며, 플레이어는 움직이지 못하는 상태가 된다. 또한 Q를 Hold한 시간에 따라 썬더볼트의 투사체의 개수가 최대 10개 까지 늘어난다. 피카츄 스킬의 대기 시간은 피카츄 머리 위에 생성된다.



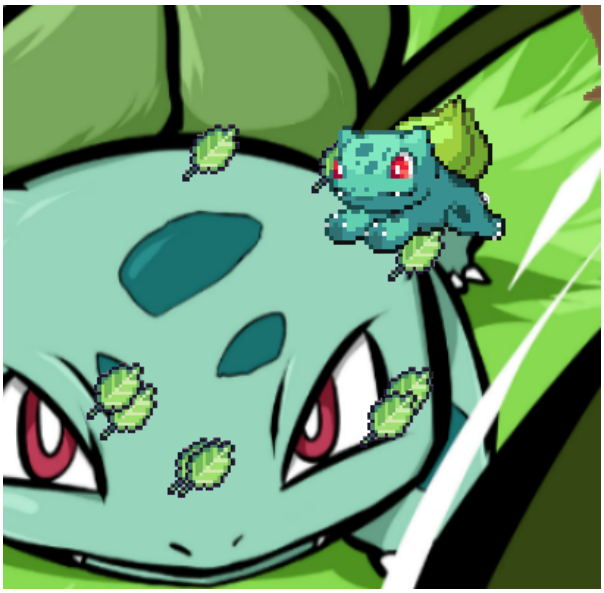


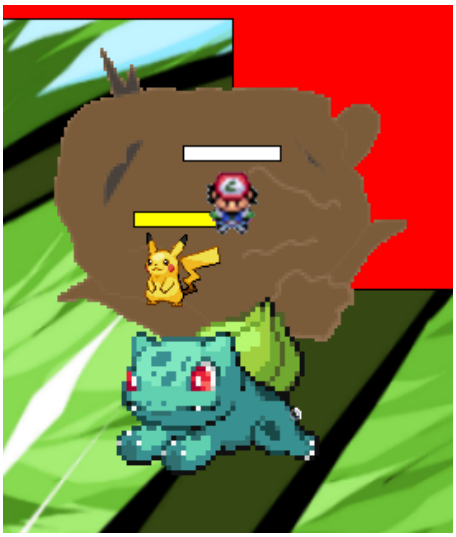
피카츄의 100만볼트는 R키를 눌러 사용되며, 재사용시간이 존재하지 않는다. 100만볼트를 사용하게 되면 썬더볼트와 같은 기능을 하는 투사체 200개가 피카츄 주변으로 바로 생성되어 퍼지게 된다. 처음에는 1개가 주어지지만 플레이어가 죽으면 1개씩 보충되어 최대 3개를 보유할 수 있으며 우측 상단에 사용 가능한 개수가 표시된다.



몬스터

몬스터는 점령되지 않은 땅에서 돌아다니면서 투사체를 발사한다. 몬스터나 투사체가 점령 중인 라인을 충돌하게 되면 플레이어는 시작 자리로 돌아가게 된다. 몬스터는 플레이어 위치에 늪지대를 생성하여 플레이어 이동속도를 저하시킨다.





2. 개발 내용

1. **구조**

핵심 로직

1. **싱글톤**

- CORE
  - 모든 매니저를 생성 및 업데이트 하는 싱글톤

```
void CCore::Progress()
{
  //Manager Update
  CTimeMgr::GetInst()->Update();
  CKeyMgr::GetInst()->Update();

  //SceneUpdate
  CSceneMgr::GetInst()->Update();
  CCollisionMgr::GetInst()->Update();

  //RENDERING
  Rectangle(memDC, -1, -1, ptResolution.x + 1, ptResolution.y + 1);

  CSceneMgr::GetInst()->Render(memDC);
  BitBlt(hDC, 0, 0, ptResolution.x, ptResolution.y, memDC, 0, 0, SRCCOPY);

  //EventManager Update
  CEventMgr::GetInst()->Update();
}
```

- EVENT
  - 게임에서 이벤트 발생(객체 생성, 삭제, Scene전환 등)시 프레임 중간에 추가/삭제 방지

```
void CEventMgr::Update()
{
  // >> : 이전 프레임 dead처리한거 삭제
  for (size_t i = 0; i < vecDead.size(); i++)
  {
    if (vecDead[i]->eGroup >= (GROUP_TYPE)0 && vecDead[i]->eGroup < GROUP_TYPE::END)
      delete vecDead[i];
  }
  vecDead.clear();
  // << : 이전 프레임 dead처리한거 삭제


  // >> : Event Execute
  for (size_t i = 0; i < vecEvent.size(); i++)
  {
    Execute(vecEvent[i]);
  }
  vecEvent.clear();
  // << : Event Execute
}

void CEventMgr::Execute(const tEvent& _eve)
{
  switch (_eve.eEvent)
```

```
  {
  case EVENT_TYPE::CREATE_OBJECT:
  {
    //  lParam : Object Address
    //  wParam: GROUP_TYPE
    CObject* pNewObj = (CObject*)_eve.lParam;
    GROUP_TYPE pNewGT = (GROUP_TYPE)_eve.wParam;
    CSceneMgr::GetInst()->GetCurrentScene()->AddObject(pNewObj, pNewGT);
  }
    break;
  case EVENT_TYPE::DELETE_OBJECT:  // Objectfmf Dead 상태로 변경 -> 한프레임 이후 실제 삭제
  {
    // lParam : Object Address
    CObject* pDeadObj = (CObject*)_eve.lParam;
    pDeadObj->setDead();
    vecDead.push_back(pDeadObj);
  }
    break;
  case EVENT_TYPE::SCENE_CHANGE:
    // lParam : Next Scene Type
    CSceneMgr::GetInst()->ChangeScene((SCENE_TYPE)_eve.lParam);
    break;
  case EVENT_TYPE::END:
    break;
  }
}
```

- SCENE
  - 현재 Scene을 포인터로 받아 Update와 Render 실행 및 Scene전환

```
void CSceneMgr::ChangeScene(SCENE_TYPE _scene)
{
  pCurScene->Exit();

  pCurScene = arrScene[(int)_scene];

  pCurScene->Enter();
}
void CSceneMgr::Update()
{
  pCurScene->Update();
  pCurScene->FinalUpdate();
}

void CSceneMgr::Render(HDC _dc)
{
  pCurScene->Render(_dc);
}
```

- COLLISION
  - Tag로 충돌 여부 확인 및 충돌 시점(첫 충돌, 충돌 중, 충돌 끝) 검사

```
void CCollisionMgr::CollisionGroupUpdate(GROUP_TYPE _eLeft, GROUP_TYPE _eRight)
{
  CScene* pCurScene = CSceneMgr::GetInst()->GetCurrentScene();

  const vector<CObject*>& vecLeft = pCurScene->GetGroupObject(_eLeft);
```

```
const vector<CObject*>& vecRight = pCurScene->GetGroupObject(_eRight);

map<ULONGLONG, bool>::iterator iter;

for (size_t i = 0; i < vecLeft.size(); ++i)
{
  if (vecLeft[i]->getCollider() == nullptr) continue; //충돌체 없음
  for (size_t j = 0; j < vecRight.size(); ++j)
  {
    if (vecRight[j]->getCollider() == nullptr)continue; //충돌체 없음
    if (vecLeft[i] == vecRight[j]) continue; //충돌체가 자기 자신

    CCollider* pLeftcol = vecLeft[i]->getCollider();
    CCollider* pRightcol = vecRight[j]->getCollider();

    //두 충돌체 조합 아이디 생성
    COLLIDER_ID ID;
    ID.Left_id = pLeftcol->getID();
    ID.Right_id = pRightcol->getID();

    iter = mapColInfo.find(ID.ID);

    if (mapColInfo.end() == iter) //이전 프레임에 충돌 안했다. -> 충돌여부 미등록
    {
      mapColInfo.insert(make_pair(ID.ID, false));
      iter = mapColInfo.find(ID.ID);
    }

    if (IsCollision(pLeftcol, pRightcol)) //현재 충돌 if
    {
      if (iter->second) //전 프레임에도 충돌 했다
      {
        // 충돌중 하나가 죽음 -> 충돌하지 않은상태로 세팅
        if (vecLeft[i]->IsDead() || vecRight[j]->IsDead())
        {
          pLeftcol->OnCollisionExit(pRightcol);
          pRightcol->OnCollisionExit(pLeftcol);
          iter->second = false;
        }
        else
        {
          pLeftcol->OnCollision(pRightcol);
          pRightcol->OnCollision(pLeftcol);
        }
      }
      else //첫 충돌
      {
        //둘 중 하나라도 죽지 않은 상태이면
        if (!vecLeft[i]->IsDead() && !vecRight[j]->IsDead())
        {
          pLeftcol->OnCollisionEnter(pRightcol);
          pRightcol->OnCollisionEnter(pLeftcol);
          iter->second = true;
        }
      }
    }
    else //충돌하지 않음
    {
      if (iter->second)
      {
        pLeftcol->OnCollisionExit(pRightcol);
        pRightcol->OnCollisionExit(pLeftcol);
        iter->second = false;
      }
      else
```

```
        {
          //전프레임에도, 지금도 충돌하지 않음
        }
      }
    }
  }

}

bool CCollisionMgr::IsCollision(CCollider* _pLeftCol, CCollider* _pRightCol)
{
  Vec2 vLeftPos = _pLeftCol->getFinalPos();
  Vec2 vLeftScale = _pLeftCol->getScale();

  Vec2 vRightPos = _pRightCol->getFinalPos();
  Vec2 vRightScale = _pRightCol->getScale();

  if (abs(vRightPos.x - vLeftPos.x) < (vLeftScale.x + vRightScale.x) / 2.f
    && abs(vRightPos.y - vLeftPos.y) < (vLeftScale.y + vRightScale.y) / 2.f)
  {
    return true;
  }

  return false;
}
```

객체

모든 객체는 Obejct 부모 클래스로부터 파생되었으며, 각 객체에서 Update와 Render을 실행한다.

- Player
  - Player은 플레이어의 입력과 이동, 점령중인 포인트들을 담당한다.
  - 이동
    - 플레이어 위치에 4방향으로 점령된 선이 있는지 확인 후 이동 가능 여부를 확인한다.

```
void CPlayer::ConfigOCDDir()
{
  OCDPoint* ocdPoint= GetOCDVec2();
  OCDPoint* temp = ocdPoint;
  Point pCenter = vCenter.ToPoint();
  for (int i = 0; i < (int)DIRECTION::END; i++)
  {
    OCDDir[i] = nullptr;
  }
  do
  {
    // >> : ON POINT
    if (ocdPoint->pPoint==pCenter)
    {
      if (pCenter.x == ocdPoint->nxt->pPoint.x)
      {
        if (pCenter.y > ocdPoint->nxt->pPoint.y)
          OCDDir[(int)DIRECTION::UP] = ocdPoint->nxt;
        else OCDDir[(int)DIRECTION::DOWN] = ocdPoint->nxt;
      }
      else if (pCenter.y == ocdPoint->nxt->pPoint.y)
      {
```

```
        if (pCenter.x < ocdPoint->nxt->pPoint.x)
          OCDDir[(int)DIRECTION::RIGHT] = ocdPoint->nxt;
        else OCDDir[(int)DIRECTION::LEFT] = ocdPoint->nxt;
      }
      if (pCenter.x == ocdPoint->prv->pPoint.x)
      {
        if (pCenter.y > ocdPoint->prv->pPoint.y)
          OCDDir[(int)DIRECTION::UP] = ocdPoint->prv;
        else OCDDir[(int)DIRECTION::DOWN] = ocdPoint->prv;
      }
      else if (pCenter.y == ocdPoint->prv->pPoint.y)
      {
        if (pCenter.x < ocdPoint->prv->pPoint.x)
          OCDDir[(int)DIRECTION::RIGHT] = ocdPoint->prv;
        else OCDDir[(int)DIRECTION::LEFT] = ocdPoint->prv;
      }
    }
    // << : ON POINT
    // >> : BETWEEN POINT
    else if (IsBetweenX(ocdPoint->pPoint, ocdPoint->nxt->pPoint, pCenter))
    {
      Vec2 newVec1 = Vec2(ocdPoint->pPoint.x, ocdPoint->pPoint.y - 10);
      Vec2 newVec2 = Vec2(ocdPoint->pPoint.x, ocdPoint->pPoint.y + 10);
      if (IsBetweenY(newVec1, newVec2, pCenter))
      {
        if (pCenter.x > ocdPoint->pPoint.x)
        {
          OCDDir[(int)DIRECTION::LEFT] = ocdPoint;
          OCDDir[(int)DIRECTION::RIGHT] = ocdPoint->nxt;
        }
        else
        {
          OCDDir[(int)DIRECTION::LEFT] = ocdPoint->nxt;
          OCDDir[(int)DIRECTION::RIGHT] = ocdPoint;
        }
      }
    }
    else if (IsBetweenY(ocdPoint->pPoint, ocdPoint->nxt->pPoint, pCenter))
    {
      Vec2 newVec1 = Vec2(ocdPoint->pPoint.x-10, ocdPoint->pPoint.y);
      Vec2 newVec2 = Vec2(ocdPoint->pPoint.x + 10, ocdPoint->pPoint.y);
      if (IsBetweenX(newVec1, newVec2, pCenter))
      {
        if (pCenter.y >= ocdPoint->pPoint.y)
        {
          OCDDir[(int)DIRECTION::UP] = ocdPoint;
          OCDDir[(int)DIRECTION::DOWN] = ocdPoint->nxt;
        }
        else
        {
          OCDDir[(int)DIRECTION::UP] = ocdPoint->nxt;
          OCDDir[(int)DIRECTION::DOWN] = ocdPoint;
        }
      }
    }
    // << : BETWEEN POINT
    ocdPoint = ocdPoint->nxt;
  } while (ocdPoint != temp);
}
```

- 플레이어 위치가 점령된 위치인지 확인
  - 점령 위치 여부에 따라 이동시 점령된 위치, 점령 중 상태를 변경한다.

```
bool CPlayer::IsPlayerOnOCD()
{
  pCenter = vCenter.ToPoint();
  OCDPoint* ocdCrnt = GetOCDVec2();
  OCDPoint* temp = ocdCrnt;

  for (int i = 0; i < (int)DIRECTION::END; i++)
  {
    if (OCDDir[i] == nullptr) continue;
      if (pCenter.x  == OCDDir[i]->pPoint.x)
      return true;
    else if (pCenter.y == OCDDir[i]->pPoint.y)
      return true;
  }
  return false;
}
```

- 점령중 Point 모으기
  - 점령 중 상태가 되면 방향을 전환할 때 마다 모은다.

```
void CPlayer::CollectOCGPoints( )
{
  if (eDirBefore != eDirCrnt && eDirBefore != OppositeDirection(eDirCrnt))
  {
    pOCGPoint[ocgCnt] = vCenter;
    if (abs(pOCGPoint[ocgCnt].x - pOCGPoint[ocgCnt - 1].x) == 1)
    {
      pOCGPoint[ocgCnt].x = pOCGPoint[ocgCnt - 1].x;
    }
    else if (abs(pOCGPoint[ocgCnt].y - pOCGPoint[ocgCnt - 1].y) == 1)
    {
      pOCGPoint[ocgCnt].y = pOCGPoint[ocgCnt - 1].y;
    }
    ocgCnt++;
  }
  else
  {
    pOCGPoint[ocgCnt] = vCenter.ToPoint();
  }
}
```

- 점령 중 상태에서 다시 안전 지역으로 돌아오면 점령한 Point들을 Ocuppied 객체(점령 된 땅 관리)에 필요 요소 등를 포장 후 전송한다.

```
if (IsPlayerOnOCD())
    {

      eState = PLAYER_STATE::ON_OCCUPIED;

      OCDPoint* ocdCrnt = GetOCDVec2();
      OCDPoint* temp = ocdCrnt;
      do
      {
        if (IsBetweenX(ocdCrnt->pPoint, ocdCrnt->nxt->pPoint, pOCGPoint[ocgCnt]))
        {
          if (abs(ocdCrnt->pPoint.y - pOCGPoint[ocgCnt].y) <= 1
            && abs(ocdCrnt->pPoint.y - pOCGPoint[ocgCnt].y) >= 0)
```

```
            pOCGPoint[ocgCnt].y = ocdCrnt->pPoint.y;
        }
        if (IsBetweenY(ocdCrnt->pPoint, ocdCrnt->nxt->pPoint, pCenter))
        {
          if (abs(ocdCrnt->pPoint.x - pOCGPoint[ocgCnt].x) <= 1
            && abs(ocdCrnt->pPoint.x - pOCGPoint[ocgCnt].x) >= 0)
            pOCGPoint[ocgCnt].x = ocdCrnt->pPoint.x;
        }
      } while (ocdCrnt != temp);
        ocgCnt++;

      CScene* crntScene = CSceneMgr::GetInst()->GetCurrentScene();
      crntScene->GetOCD()->FinishOCG(GetOCGPack());
      OCGonOCD = false;
      ocgCnt = 0;
    }


//포장 함수
PackOCG CPlayer::GetOCGPack()
{
  bool bSameDir = eDirStart == eDirCrnt ? true : false;
  if (pOCGPoint[ocgCnt - 1] == Point(0, 0)) ocgCnt--;
  if (pOCGPoint[0] == Point(0, 0))
  {
    for (int i = 1; i < ocgCnt; i++)
    {
      pOCGPoint[i - 1] = pOCGPoint[i];
    }
  }
  return PackOCG(pOCGPoint, ocgCnt, IsClockwise(), eDirStart, bSameDir);
}
```

- **Pokemon**
  - 플레이어를 따라다니며 입력키를 받아 스킬을 실행한다
- 플레이어 따라다니기

```
void CPokemon::FollowPosition()
{
  int radius = 40;

  if (GetDistance(vPlayer, vCenter) > radius)
  {
    vCenter = vPlayer;
    switch (eDirPlayer)
    {
      case DIRECTION::UP:
        vCenter.y += radius;
        vCenter.x += 15;
        break;
      case DIRECTION::DOWN:
        vCenter.y -= radius;
        vCenter.x += 15;
        break;
      case DIRECTION::RIGHT:
        vCenter.x -= radius;
        break;
      case DIRECTION::LEFT:
```

```
        vCenter.x += radius;
        break;
    }
  }
}
```

- 썬더볼트 생성

```
void CPokemon::ThunderBolt()
{
  for (int i = 0; i < iBoltQuant; i++)
  {
    Vec2 vVectorTo = Vec2(rand() % 200 - 100, rand() % 200 - 100);
    float speed = rand() % 30 + 40;
    CreateThunderBolt(vVectorTo, speed);
  }
}

void CPokemon::CreateThunderBolt(Vec2 _dir, float _fspeed)
{
  CMissile* pMissile = new CMissile();
  pMissile->SetPosition(vCenter);
  pMissile->InitThunderBolt(_dir, _fspeed);

  CreateObject(pMissile, GROUP_TYPE::PLAYER_PROJECTILE);
}
```

- 100만볼트 생성

```
void CPokemon::Ultimate()
{
  CMissile* pMissile = new CMissile();

  bool bDoOnce = false;
  for (int i = 0; i < 100; i++)
  {
    pMissile = new CMissile();
    pMissile->PikachuUltimate(vCenter);
    CreateObject(pMissile, GROUP_TYPE::PLAYER_PROJECTILE);
  }
  iUltimate--;
  CScene* crntScene = CSceneMgr::GetInst()->GetCurrentScene();
  crntScene->PokemonUseUltimate(iUltimate);
}
```

- 몬스터
  - 맵을 이동하면서 플레이어게 투사체 공격 한다.
  - 이동

```
void CEnemy::EnemyIdle()
{
  eState = ENEMY_STATE::IDLE;
  //vDir = Vec2(rand() % 200 - 100, rand() % 200 - 100);
```

```
  Vec2 vPlayer = crntScene->GetPlayer()->GetPosition();
  Vec2 newDir = Vec2(vPlayer.x - vCenter.x, vPlayer.y - vCenter.y);
  vDir = (newDir);
  if (vDir.x == 0) vDir.x = 1;
  if (vDir.y == 0) vDir.y = 1;
  vDir.Normalize();
  fIdleDuration = rand() % 5 + 3;
}
```

- 투사체 공격

```
void CEnemy::CreateRazorLeaf()
{
  fDTRazorLeaf = fDTLeaf1;
  Vec2 vVectorTo;
  for (int i = 0; i < 10; i++)
  {
    vVectorTo = Vec2(rand() % 200 - 100, rand() % 200 - 100);
    CreateLeaf1(vVectorTo, 10);
  }
}
```

- 늪지대 공격

```
void CEnemy::CreateSwamp()
{
  CMissile* pMissile = new CMissile();
  pMissile->SpreadSwamp();

  CreateObject(pMissile, GROUP_TYPE::PLAYER_DEBUFF);
  fSwampCycle = 10 + rand() % 5;
}
```

- 몬스터와 점령 중 선과 충돌

```
void CEnemy::HitOCG()
{
  CScene* crntScene = CSceneMgr::GetInst()->GetCurrentScene();
  PackOCG packOCG = crntScene->GetPlayer()->GetOCGPack();
  Point pCenter = vCenter.ToPoint();
  Point pScale = GetScale();
  int radius =50;
  for (int i = 0; i < packOCG.ocgCnt; i++)
  {
    if (IsBetweenY(packOCG.ocgPoint[i], packOCG.ocgPoint[i + 1], pCenter))
    {
      if (IsBetweenTwo(packOCG.ocgPoint[i].x + radius, packOCG.ocgPoint[i].x - radius, pCenter.x))
        crntScene->GetPlayer()->GoBackOCG0();
    }
    if (IsBetweenX(packOCG.ocgPoint[i], packOCG.ocgPoint[i + 1], pCenter))
    {
      if (IsBetweenTwo(packOCG.ocgPoint[i].y + radius, packOCG.ocgPoint[i].y - radius, pCenter.y))
        crntScene->GetPlayer()->GoBackOCG0();
    }
  }
}
```

- 투사체
  - 몬스터, 포켓몬의 투사체 생성, 발사 및 충돌 감지
  - 몬스터 투사체와 점령 중 선과 충돌

```
void CMissile::HitOCG()
{
  CScene* crntScene = CSceneMgr::GetInst()->GetCurrentScene();
  PackOCG packOCG = crntScene->GetPlayer()->GetOCGPack();
  Point pCenter = vCenter.ToPoint();
  Point pScale = GetScale();
  pCenter.x += pScale.x / 2;
  pCenter.y += pScale.y / 2;
  for (int i = 0; i < packOCG.ocgCnt; i++)
  {
    if (IsBetweenY(packOCG.ocgPoint[i], packOCG.ocgPoint[i + 1], pCenter))
    {
      if(IsBetweenTwo(packOCG.ocgPoint[i].x+2, packOCG.ocgPoint[i].x-2,pCenter.x))
        crntScene->GetPlayer()->GoBackOCG0();
    }
    if (IsBetweenX(packOCG.ocgPoint[i], packOCG.ocgPoint[i + 1], pCenter))
    {
      if (IsBetweenTwo(packOCG.ocgPoint[i].y + 2, packOCG.ocgPoint[i].y - 2, pCenter.y))
        crntScene->GetPlayer()->GoBackOCG0();
    }
  }
}
```

  - 타 객체 (몬스터, 플레이어 투사체, 몬스터 투사체)와 충돌

```
void CMissile::OnCollisionEnter(CCollider* _pOther)
{
  //player projectile
  if (_pOther->getOwnerObject()->getGroup() == GROUP_TYPE::MONSTER)
  {
    myDeleteObject(this, GROUP_TYPE::PLAYER_PROJECTILE);
  }
  if(eType==MISSILE_TYPE::LAZORLEAFT)
  {
    if (_pOther->getOwnerObject()->getGroup() == GROUP_TYPE::PLAYER_PROJECTILE)
    {
      myDeleteObject(this, GROUP_TYPE::MONSTER_PROJECTILE);
    }
  }
  if(eType==MISSILE_TYPE::THUNDERBOLT||eType==MISSILE_TYPE::PIKACHU_ULTIMATE)
  {
    if (_pOther->getOwnerObject()->getGroup() == GROUP_TYPE::MONSTER_PROJECTILE)
    {
      myDeleteObject(this, GROUP_TYPE::PLAYER_PROJECTILE);
    }
  }
}
```

- 점령된 땅

점령된 땅의 Point들이 필요한 객체에 지속적으로 Point들을 제공하고 점령이 완료되면 점령된 땅을 업데이트 한다.

- 점령이 완료되면 원형 리스트에 점령한 Point를 업데이트 한다.
- 점령한 Point안에 몬스터가 있는지 확인

```
bool COcuppied::IsEnemyInOCG()
{
  int enemyLineCnt = 0;

  //몬스터 좌우위아래 중 ocg 선이 홀수면 in 짝수면 out
  DIRECTION startCnt = DIRECTION::END;
  for (int i = 0; i < ocgPack.ocgCnt; i++)
  {
    if ((startCnt == DIRECTION::END || startCnt == DIRECTION::RIGHT) && vEnemyCen.x < ocgPack.ocgPoint[i].x)
    {
      if (IsBetweenY(ocgPack.ocgPoint[i], ocgPack.ocgPoint[(i + 1)%ocgPack.ocgCnt], vEnemyCen))
      {
        enemyLineCnt++;
        startCnt = DIRECTION::RIGHT;
      }
    }
    if ((startCnt == DIRECTION::END || startCnt == DIRECTION::LEFT) && vEnemyCen.x > ocgPack.ocgPoint[i].x)
    {
      if (IsBetweenY(ocgPack.ocgPoint[i], ocgPack.ocgPoint[(i + 1) % ocgPack.ocgCnt], vEnemyCen))
      {
        enemyLineCnt++;
        startCnt = DIRECTION::LEFT;
      }
    }
    if ((startCnt == DIRECTION::END || startCnt == DIRECTION::DOWN) && vEnemyCen.y < ocgPack.ocgPoint[i].y)
    {
      if (IsBetweenX(ocgPack.ocgPoint[i], ocgPack.ocgPoint[(i + 1) % ocgPack.ocgCnt], vEnemyCen))
      {
        enemyLineCnt++;
        startCnt = DIRECTION::DOWN;
      }
    }
    if ((startCnt == DIRECTION::END|| startCnt == DIRECTION::UP) && vEnemyCen.y > ocgPack.ocgPoint[i].y)
    {
      if (IsBetweenX(ocgPack.ocgPoint[i], ocgPack.ocgPoint[(i + 1) % ocgPack.ocgCnt], vEnemyCen))
      {
        enemyLineCnt++;
        startCnt = DIRECTION::UP;
      }
    }
  }
  if (enemyLineCnt % 2 != 0)
  {
    return true;
  }
  return false;
}
```

- 삭제 시작점과 끝점, 추가 시작점을 확인한다.

```
void COcuppied::ConfigDeletePoint()
{
  ocdDeleteStart = nullptr;
```

```
  ocdDeleteEnd = nullptr;
  ocdAddStart = nullptr;
  OCDPoint* temp = ocdVec2;
  if (ocgPack.bClockwise)
  {
    do
    {
      if (IsOnXorYAxis(ocdVec2->pPoint, ocdVec2->nxt->pPoint, ocgPack.ocgPoint[0])
        &&ocdDeleteStart==nullptr)
        //|| ocdVec2->pPoint.AroundEqual(ocgPack.ocgPoint[0]))
      {
        ocdDeleteStart = ocdVec2->nxt;
        ocdAddStart = ocdVec2;
      }
      if (IsOnXorYAxis(ocdVec2->pPoint, ocdVec2->nxt->pPoint, ocgPack.ocgPoint[ocgPack.ocgCnt - 1]))
        //|| ocdVec2->pPoint.AroundEqual(ocgPack.ocgPoint[ocgPack.ocgCnt - 1]))
      {
        ocdDeleteEnd = ocdVec2;
      }
      if (ocdVec2->pPoint.AroundEqual(ocgPack.ocgPoint[0]))
      {
        ocdDeleteStart = ocdVec2;
        ocdAddStart = ocdVec2->prv;
      }
      ocdVec2 = ocdVec2->nxt;
    } while (temp != ocdVec2);
  }
  else //ocgPack.bClockwise == true
  {
    do
    {
      if (IsOnXorYAxis(ocdVec2->pPoint, ocdVec2->nxt->pPoint, ocgPack.ocgPoint[0])
        &&ocdDeleteStart==nullptr)
        //|| ocdVec2->pPoint.AroundEqual(ocgPack.ocgPoint[0]))
      {
        ocdDeleteStart = ocdVec2;
        ocdAddStart = ocdVec2->nxt;
      }
      if (IsOnXorYAxis(ocdVec2->pPoint, ocdVec2->nxt->pPoint, ocgPack.ocgPoint[ocgPack.ocgCnt - 1]))
        //|| ocdVec2->pPoint.AroundEqual(ocgPack.ocgPoint[ocgPack.ocgCnt - 1]))
      {
        ocdDeleteEnd = ocdVec2->nxt;
      }
      if (ocdVec2->pPoint.AroundEqual(ocgPack.ocgPoint[0]))
      {
        ocdDeleteStart = ocdVec2->prv;
        ocdAddStart = ocdVec2;
      }
      ocdVec2 = ocdVec2->nxt;
    } while (temp != ocdVec2);
  }
}
```

- 삭제 시작점부터 끝점까지 Point들을 삭제한다.

```
void COcuppied::DeleteOCDPoint()
{
  OCDPoint* temp = ocdDeleteStart;
  //시작점과 끝점 사이에 아무 점이 없고 몬스터가 안에 없을 때
  if (ocdDeleteStart == ocdDeleteEnd&&!ocdDeleteStart->pPoint.AroundEqual(ocdDeleteEnd->pPoint))
  {
    ocdDeleteStart->DeleteSelf();
```

```
    ocdDeleteStart = nullptr;
  }
  //시작점과 끝점 사이에 아무 점이 없고 몬스터가 안에 있을 때
  else if (IsEnemyInOCG()&&(ocdDeleteStart == ocdAddStart || ocdDeleteEnd == ocdAddStart))
  {
    if(ocgPack.bClockwise)
    {
      while (temp != ocdDeleteEnd)
      {
        temp = ocdDeleteStart->nxt;
        ocdDeleteStart->DeleteSelf();
        ocdDeleteStart = temp;
      }
      ocdDeleteEnd->DeleteSelf();
    }
    else //ocgPack.bClockwise==false;
    {
      while (temp != ocdDeleteEnd)
      {
        temp = ocdDeleteStart->prv;
        ocdDeleteStart->DeleteSelf();
        ocdDeleteStart = temp;
      }
      ocdDeleteEnd->DeleteSelf();
    }

    Sannabi = true;
  }
  else if (ocgPack.bClockwise)
  {
    OCDPoint* ocdEndNxt = ocdDeleteEnd->nxt;
    while(temp!=ocdEndNxt)
    {
      temp = temp -> nxt;
      ocdDeleteStart->DeleteSelf();
      ocdDeleteStart = temp;
    }
  }
  else //ocgPack.bClockwise==false
  {
    OCDPoint* ocdEndPrv = ocdDeleteEnd->prv;
    while (temp != ocdEndPrv)
    {

      temp = temp->prv;
      ocdDeleteStart->DeleteSelf();
      ocdDeleteStart = temp;
    }
  }
}
```

- 추가 시작점에 Point들을 추가한다

```
void COcuppied::AddOCGPoint()
{
  int i = 0;

  if (bStartOCGonOCD) i = 1;
  if (bEndOCGonOCD) ocgPack.ocgCnt--;
  if (!Sannabi)
  {
    if (ocgPack.bClockwise)
```

```
    {
      for (; i < ocgPack.ocgCnt; i++)
      {
        ocdAddStart->InsertNxt(ocgPack.ocgPoint[i].ToVec2());
        ocdAddStart = ocdAddStart->nxt;
      }
    }
    else //ocg.PackbClockwise==false
    {
      for (; i < ocgPack.ocgCnt; i++)
      {
        ocdAddStart->InsertPrv(ocgPack.ocgPoint[i].ToVec2());
        ocdAddStart = ocdAddStart->prv;
      }
    }
    ocdVec2 = ocdAddStart;
  }
  else
  {
    ocdVec2= new OCDPoint(ocgPack.ocgPoint[2]);
    ocdVec2->prv = new OCDPoint(ocgPack.ocgPoint[1]);
    ocdVec2->nxt = ocdVec2->prv;
    ocdVec2->prv->prv = ocdVec2;
    ocdVec2->prv->nxt = ocdVec2;

    ocdVec2->InsertNxt(ocgPack.ocgPoint[0]);

    for (int i = 3; i < ocgPack.ocgCnt; i++)
    {
      ocdVec2->InsertNxt(ocgPack.ocgPoint[i].ToVec2());
      ocdVec2 = ocdVec2->nxt;
    }

    Sannabi = false;
  }
}
```

- 점령이 완료되면 위 함수들을 토대로 점령된 땅을 업데이트 한다.

```
void COcuppied::FinishOCG(PackOCG _ocgPack)
{
  ocgPack = _ocgPack;
  CScene* crntScene = CSceneMgr::GetInst()->GetCurrentScene();
  vEnemyCen = crntScene->GetEnemy()->GetPosition();

  if(!ocgPack.bSameDir)
  {
    if (!IsEnemyInOCG())
      ocgPack.bClockwise = !ocgPack.bClockwise;
    ConfigDeletePoint();
  }
  else
  {
    OCGSameDirection();
  }
  if (ocdDeleteStart == nullptr || ocdDeleteEnd == nullptr || ocdAddStart == nullptr) return;

  DeleteOCDPoint();
  AddOCGPoint();

  bStartOCGonOCD = false;
  bEndOCGonOCD = false;
```

```
  StraightOCD();
  UpdateOCDArea();
  if (fTotalPercent> FinishRatio)
  {
    bFinishGame = true;
    fDTFinishGame = 0;
  }
}
```